Acquisition, Representation and Rule Generation for Procedural Knowledge

Chris Ortiz
Software Technology Branch/PT4, NASA/Johnson Space Center, Houston, Texas 77058.
(ortiz@gothamcity.jsc.nasa.gov)

Tim Saito
Computer Sciences Corporation, 16511 Space Center Boulevard - M30, Houston, Texas 77058, U.S.A.
(tsaito@nasamail.nasa.gov)

Sachin Mithal
Computer Sciences Corporation, 16511 Space Center Boulevard - M30, Houston, Texas 77058,
(sachin@gothamcity.jsc.nasa.gov)

R. Bowen Loftin*
Department of Natural Sciences, University of Houston-Downtown, One Main Street, Room 813-N, Houston, Texas 77002, U.S.A., and NASA/Johnson Space Center
(bloftin@nasamail.nasa.gov)

Historically knowledge acquisition has proven to be one of the greatest barriers to the development of intelligent systems. Current practices generally require lengthy interactions between the expert whose knowledge is to be captured and the knowledge engineer whose responsibility is to acquire and represent the expert's knowledge in a useful form. Although much research has been devoted to the development of methodologies and computer software to aid in the capture and representation of some types of knowledge, little attention has been devoted to procedural knowledge. NASA personnel, on the other hand, frequently perform tasks that are primarily procedural in nature.

This paper describes current research into the design and continuing development of a system for the acquisition of procedural knowledge, its representation in useful forms, and proposed methods for automated CLIPS rule generation. TARGET (Task Analysis and Rule Generation Tool) is intended to permit experts, individually or collectively, to visually describe and refine procedural tasks. The system is designed to represent the acquired knowledge in the form of graphical objects with the capability for generating production rules in CLIPS. The generated rules can then be integrated into applications such as NASA's ICAT (Intelligent Computer Aided Training) architecture. The paper concludes by describing proposed methods for use in translating the graphical and intermediate knowledge representations into CLIPS rules.

Systems such as TARGET have the potential to profoundly reduce the time, difficulties, and costs of developing knowledge-based systems for the performance of procedural tasks.

INTRODUCTION

Processes and software designed to aid knowledge acquisition can be characterized by the nature of their delivery and implementation methods and styles as well as their ability to extract knowledge. Various authoring tools have evolved to solve the problems associated with the creation of a specific expert system

* Address all inquiries to R. B. Loftin, Mail Code PT4, NASA/Johnson Space Center, Houston, TX 77058, (713) 483-8070, bloftin@nasamail.nasa.gov.

(Boose, 1989). Historically, most knowledge-acquisition-oriented tool designs were directed toward rating or categorizing problems or knowledge. To use such tools to capture specific knowledge, the developer distinguished between types of knowledge methods/approaches. Although sharing many of the same goals, the existing methodologies are numerous, ranging from frame modeling to case-based reasoning models to repertory-grid rating structures. The various knowledge types addressed by these systems—from semantic/taxonomic to declarative to procedural—affect the design and performance decisions of researchers and implementers (Gaines, 1988). Knowledge representations, including frames, objects, rules, and decision trees, are used to capture and execute expertise. At this point, most would agree that no one tool accommodates all of the cognitive styles needed to gather the information/knowledge necessary for the creation of an expert system in one contiguous process. It is clear that viable standards have yet to be fully established and accepted.

Procedural knowledge acquisition via task analysis is a reasonable candidate for graphical representation modes. Decomposing a complex set of steps that make up a specific mission or task requires cognitive visualization and the ability to formulate and reformulate the decomposition of those steps or actions. The specific heuristic procedures that most subject matter experts (SMEs) employ share certain levels of organization and recall (de Kleer, Doyle, Steele, Jr., and Sussman, 1985). The path in which a procedure evolves starts with specific agendas and goals. The last or final action of reaching or satisfying those actual goals would end the procedure. On the other hand, any actions that would restart a process (i.e., a loop) would occur before the goal-oriented or last action. Decisions may be made during a task that direct the expert along alternative paths that may or may not be taken in other performances of the same task. In cases where the processes offer one or more options to complete a task, the process diverges into as many paths as necessary to meet the optional requirements. Each path would then contain specific values for technique evaluation or other modes of feedback. These types of complexities lend themselves to representation in a visual form.

ENVIRONMENTAL FACTORS AT NASA

As in other environments, getting and maintaining an SME's attention, time, commitment, help, and data sources at NASA is usually difficult at best. SMEs tend to differ in their communication abilities and styles, willingness to cooperate, availability, and degree of computer literacy, potentially affecting the overall success of the knowledge acquisition process (Littman, 1988). The strategy of providing the SME with a tool that can be used to document his mission(s) or task(s), on his own and within his schedule, would serve to resolve some of the difficulties associated with a knowledge engineer constantly "hovering over" an SME. However, the disadvantages of such a strategy may include the lack of positive reinforcement or external motivation (i.e., SMEs might put off documenting their task/mission unless periodically reminded or encouraged).

Other constraints affecting the type of tools delivered and used could be those associated with budget problems or deficits. Where exotic workstations might be required for more sophisticated KA tools, NASA may only have dated or under-powered PC hardware in certain areas of need. Distribution of KA tools to NASA personnel who have access to inadequate equipment to use in documenting their procedures could prove frustrating. SMEs tend to use their PCs or MacIntoshes for spreadsheets, databases, or word processing where KA is not a daily issue in their operation. The need to deliver tools that do not require SMEs to alter their current work style and environment (hardware, software, operating systems, methods, etc.) is, therefore, critical.

THE CLIPS FACTOR

In the NASA/Johnson Space Center environment, CLIPS (C-Language Integrated Production System) is widely used as an expert system development and delivery vehicle. Within the ICAT metaphor the overall procedure or task is decomposed into sets of tasks/subtasks that are termed actions. For most effective use, actions are expressed, within reason, at the lowest possible level. At any point in an ICAT training session, the expert expects the trainee to perform some action. Each action, as defined by the expert, is represented as a CLIPS fact (Figure 1) in the following pattern:

(message-sender-to-receiver <step number> <action type) <argument><argument> ...)

Figure 1. CLIPS Fact for ICAT Environment

An action itself comprises at least two <argument> fields that define one single action decomposed into a hierarchical structure of two or more subactions of the form (<action> <argument>). Each <argument> may itself be an <action> at the next lower level. For example, (arg1 arg2 ... argn) can be decomposed into at least two levels: (arg1 arg2) and (arg2 . . . argn). The first pair, (arg1 arg2), is an (<action> <argument>) pair at the top level where the action arg1 has one argument, arg2. In turn, (arg2 ... argn) is another (<action> <argument>) pair at the second level where arg2 has one or more arguments, depending on the value of n. The structure for each action may be different and the number of arguments that belong with each action is variable. The expert is free to decompose the actions and arguments into hierarchies that fit his or her specific domain.

This paper details the design and implementation of a knowledge acquisition system tailored to the acquisition and representation of procedural knowledge associated with the performance of complex tasks. The primary goal of this effort has been the production of a system with an easy-to-learn and "comfortable" user interface that provides powerful mechanisms for the visual expression of procedural knowledge. The ultimate goal of this work is the expression of acquired knowledge in the form of production rules to facilitate the use of the acquired knowledge in expert systems for mission support and training.

THE TARGET (Task Analysis/Rule GEneration Tool) APPROACH

TARGET and Design Strategy

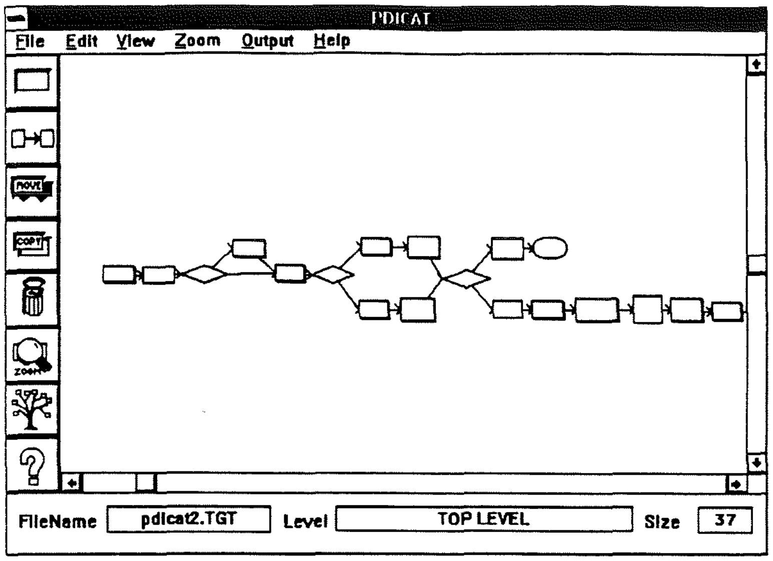
Balancing non-programmer usability, design sophistication, and hardware portability requirements, the Task Analysis/Rule GEneration Tool (TARGET) is designed to provide a knowledge acquisition environment for users of commonly-available computer systems (IBM® PCs and Apple© Macintoshes®). The forte of TARGET is the gathering of task or procedural knowledge to be expressed and analyzed graphically as well as contextually. TARGET provides users the ability to graphically decompose a task or procedure using a box-flow presentation/manipulation style within a windowed environment.

TARGET is designed to let the SMEs start documenting their job or task with minimal training in its use and no absolute need for knowledge engineer intervention. If the SME is unable to find time to work on the knowledge acquisition process alone, TARGET does allow the knowledge engineer and SME to work together in iterative sessions. It is tailored to accommodate a wide range of users, from the novice to the expert. Users can develop a discrete representation of tasks and subtasks within their domains (Payne and Green, 1986). The system then manages the information entered and represents the knowledge in a "top-down" reporting format that can then be used for rule induction and generation.

In order to support the development of intelligent computer-aided training (ICAT) systems, TARGET implements its rule representations using the rule types and structures originally developed for ICAT systems. TARGET is designed to deal with a series of tasks/subtasks that are performed procedurally or in steps. TARGET supports three action types: required, optional and flexible (as defined in the ICAT design architecture). Steps are defined as the progression from one required action to the next. Steps are represented by numerals and their values increase with the progression of the task.

TARGET User Interface

TARGET maintains a fragile balance between ease of use and design complexity/intricacy. Although it does not possess the "bells and whistles" of more sophisticated systems like Aquinas and Protege, TARGET provides enough knowledge modeling (procedural/declarative) functionality to allow the SME or knowledge engineer to build a moderately elaborate knowledge base without sacrificing the attractiveness of its user interface (Figure 2).



FileName pdicat2.TGT Level TOP LEVEL Size 37

Figure 2. TARGET Interface

TARGET provides a windowed environment through which decomposition can be organized and recorded. Ultimately the user, knowledge engineer or SME, is responsible for the overall quality checking of the knowledge base before its representation in or transfer to other applications. TARGET's report facilities offer some assistance in this quality checking process. Reports can be generated to provide moderately high-level feedback to the knowledge engineer and SME. TARGET produces the following reports:

- Task hierarchy: keeps a sequential/hierarchical account of tasks
- User Scratch Pad: keeps notes on conditions, states or other user-supplied details.

TARGET supports the identification and conceptualization phases of knowledge acquisition with its network approach to knowledge representation. Duties, tasks/subtasks, or steps/substeps within a process can be defined, documented, and structured to reflect these relationships to other duties, tasks/subtasks, or steps/substeps.

Given its developmental state, TARGET provides a reasonably comprehensive mechanism for generating simple representations at the very first knowledge acquisition session. The next sessions may be used to embellish what has already been elicited or to create new or modified versions of the knowledge base. The TARGET knowledge acquisition interface gives the user the freedom to generate as complex a hierarchy of knowledge as necessary. However, the disadvantage to such freedom is the ability to create a completely abstract knowledge base with relatively few standards for input. Some guiding controls from the TARGET interface could provide structure to the knowledge acquisition process and greatly enhance the ability of the user to create a "useful" knowledge base.

Task-Action Concepts (Building Blocks)

TARGET employs a free-form flow charting strategy. Users can explain procedural processes by the use of various flow chart icons manipulated in the work area. Tasks can then be linked together using directed arcs to represent procedural flow.

The task icons are separated into five major categories. The shape of the task box is an important key to determining its function. TARGET also works reasonably well on monochrome systems with the combination of shapes and colors.

The first category consists of the actions that are required to complete a process (Figure 3). The required actions are denoted by using a blue rectangular box on the screen. Likewise, optional tasks are represented using grey rectangles. The use of color representing various tasks has been reduced to just two, grey for optional tasks and blue for all others.

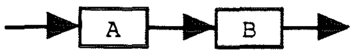


Figure 3. Required Tasks

A third task type, a hexagonal shaped box, shows where decisions are made. The connection labels reflect the choices allowed. For example, a decision box may ask if a process has been completed. This decision may branch to two other paths (Figure 4). The first arc leaving the decision may be labeled "YES" and the other labeled "NO". The arc labels represent the only possible choices allowed by the decision point. Decisions are not limited to binary operations but can have many unique arcs as necessary. The minimum number of arcs from a decision task is two.

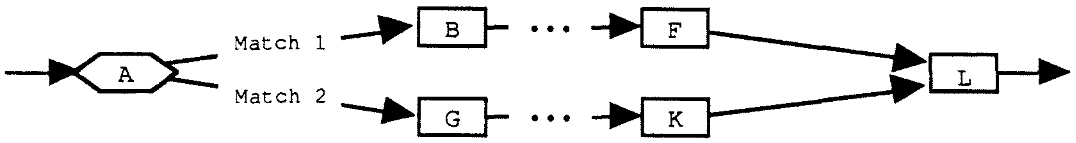


Figure 4. Decision Task

The fourth task type is a control structure. Control structures are used as a "go-to" or looping mechanism. They are ellipse-shaped to distinguish them from other tasks (Figure 5). Controls can only jump to other tasks that are on the current layer (see discussion of layers below). However, they may not jump directly to other controls. TARGET ensures that an endless loop can not exist. In addition, the control task cannot be further decomposed. TARGET will also prevent any changes to a task which a control structure is pointing to.

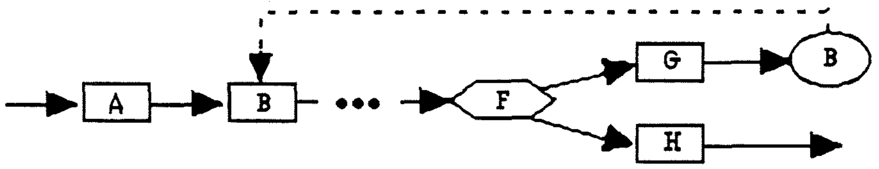


Figure 5. Control Structure

The fifth structure is a goal or an end of a process. Goal boxes are used to terminate the process or procedures. Goal tasks are denoted by using an rectangle with thick borders to distinguish them from other tasks (Figure 6). TARGET enforces certain rules regarding goals that cannot be further decomposed and may not have links originating from them.

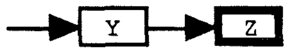


Figure. 6 Goal Structure

TARGET attempts to address the issue of parallel tasks whrere two or more sets of tasks are performed simultaneously. They are not assigned task shapes but are created whenever two or more arcs emanate from a rectangular box. Figure 7 shows a parallel task configuration. Each task chain represents a parallel process which will be executed independently until the paths converge.

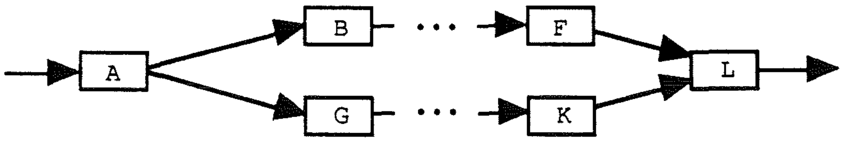
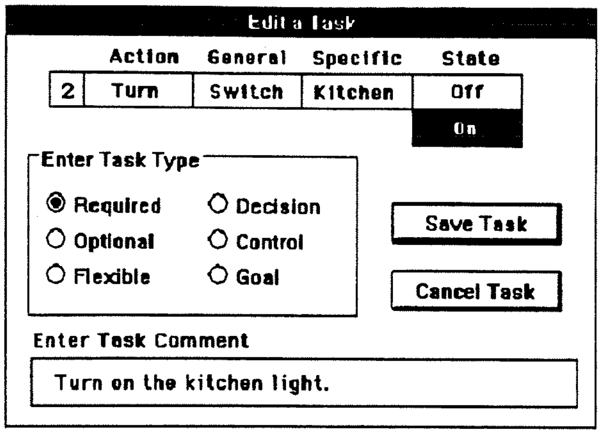


Fig. 7. Parallel Tasks

TARGET provides users with the ability to decompose various tasks into lower level tasks. Mouse manipulation makes navigating through task hierarchies fairly simple.

Example: Creating a Task

The following example will create a task which will activate a kitchen light. After selecting the create/edit icon from the toolbox and clicking on an open area in the user interface, a dialog box above will appear (Figure 8).



Edit a Task

	Action	General	Specific	State
2	Turn	Switch	Kitchen	Off
				On

Enter Task Type

[✓ Required] Optional Flexible Decision Control Goal

Save Task

Cancel Task

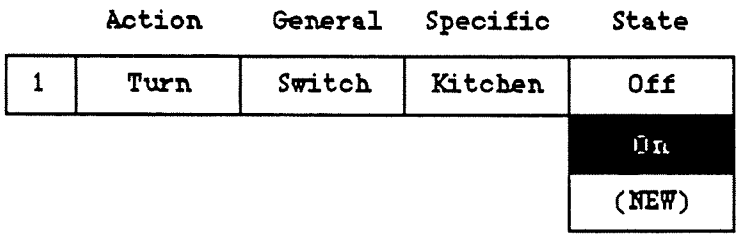
Enter Task Comment

Turn on the kitchen light.

Figure 8. Dialog Box Template

In this example, the user has entered a free form task description “Turn on the kitchen light” and has chosen “Required” as the task type. Finally information used to generate rules is entered. The rule information will assert a fact that the kitchen light has been turned on. The user may enter a number of facts that will be asserted from this dialog box. Other CLIPS functions can be used as well as user defined functions. The rule generated from this task will assert the following fact (kitchen switch turn on).

To insure that information asserted into CLIPS is consistent, a action template is provided. Entering information from the template is done from left to right. The first column represents the number of facts to be asserted. In Figure 9, the "Action" column will contain second person/present tense verbs that reflect the action within the domain. Once a verb has been selected, a "General" column is generated listing all objects associated with the selected action. Each object is then broken down further, into a list of "Specific"objects. The last column represents all valid "States" for each specific object. The user may, at anytime, insert new actions, objects or states into the network by selecting the (NEW) option.



	Action	General	Specific	State
1	Turn	Switch	Kitchen	Off
				On
				(NEW)

Figure 9. Semantic Format for Specified Action

RULE GENERATION METHODOLOGY

Knowledge Representation in CLIPS

The Rule generation component of TARGET takes the graphical description of a process and translates it into CLIPS rules. This section focuses on the graphics-to-CLIPS translation methodology using examples with CLIPS rules.

TARGET was originally designed to produce rules which could be incorporated directly into an ICAT (Intelligent Computer Aided Training) expert system architecture. In the ICAT architecture, the procedural rules interface with, and are controlled by, other CLIPS systems using a blackboard architecture. The following rule format will provide a simple control mechanism for the rules in this paper (Figure 10).

```
(defrule control_rule
                ?step <- ( next_step ?number )
=>
                (retract ?step )
                (assert (step ?number))
)
```

Figure 10. Simple Control Rule Format

The control rule acts like a traffic controller. It will receive a fact with the field next_step. The control rule will then retract this fact from the fact list and assert a new fact called step. The step fact will then trigger a TARGET rule which in turn will assert a next_step fact.

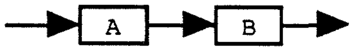
The rules, in Figure 11, produced conforms to a simple guideline. The rule will match on a control fact signifying the previous task has been executed. The rule will then retract that fact, call a series of functions needed to complete the current task, and finally assert a fact that this task has successfully completed its operation.

```
( defrule name
                ?step <- (previous task has been completed)
=>
                ( retract the previous task from the fact list )
                ( do zero or more functions (printout, assert, user defined, etc))
                ( assert a fact that this task has completed)
)
```

Figure 11. Rule Template in English

The following examples are intended to unite the TARGET graphical representation with skeletal CLIPS rules. In the following examples each task is labeled with the control facts using the same labeling approach.

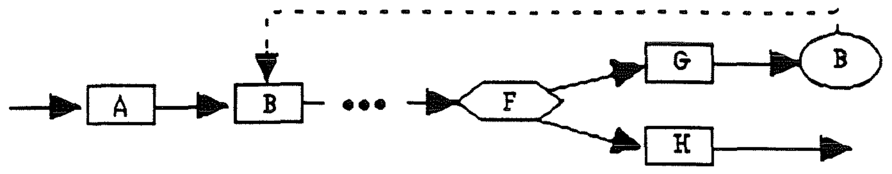
In the simplest case, all the tasks are linear. In the following example, task A must be executed before task B and so on (Figure 12). The rule generated for task B can only fire only after the previous step has completed. The control fact is then removed from the fact list. A series of functions are performed and a new control fact is asserted representing the completion of task B.



```
(defrule basic_case_B
                ?step <- (step A)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step B))
)
```

Figure 12. Sequential Tasks

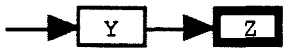
Control task rules in Figure 13 will fire only after the previous task has completed but will not process a function. They are only used to assert a control fact which will activate the task they are pointing to.



```
(defrule goto_rule
                ?step <- (step G)
=>
                (retract ?step)
                (assert next_step A)
)
```

Figure 13. Control Task Rule

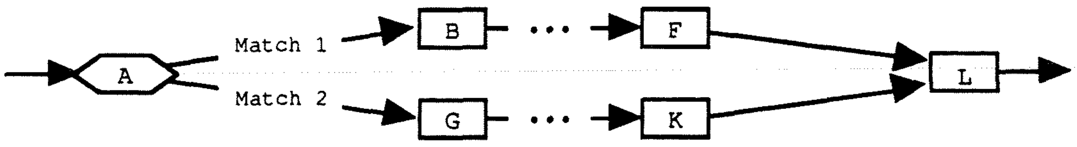
A rule derived from a goal task (Figure 14) will retract the previous control fact, process functions, but will not assert a control task, thus ending a process.



```
(defrule goal_rule_Z
                ?step <- (step Y)
=>
                (retract ?step)
                (function)
                (function)
)
```

Figure 14. Goal Task Rule

A more complex rule is generated when decisions and branching are involved. In the following example, Figure 15, decision A has two possible answers represented by Match 1 and Match 2. Depending on the outcome of the decision, one path B or G will be activated. When the two paths converge, task L must insure that either task F **or** task K has been successfully completed.



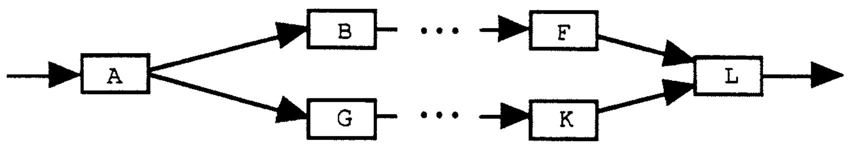
```
(defrule decision_rule_B
                ?step <- ( step A )
                (question_A match1)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step B))
)
```

```
(defrule end_decision_rule_L
                ?step <- (step F|K)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step L))
)
```

```
(defrule decision_rule_G
                ?step <- ( step A )
                (question_A match2 )
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step G))
)
```

Figure 15. Parallel Alternative Paths

The CLIPS rules generated for parallel tasks are similar to those generated by decisions but differ in a few significant ways (Figure 16). First, matching on a decision fact is not required by the rules generated for B and G. In other words, once Task A has been completed, the two rules, B and G are both activated. Secondly, in rules B and G, the control fact from the previous step is not retracted from the fact list. If the control fact was retracted from the fact list, the first rule firing would prevent other tasks from being activated. The fact will remain on the fact list until it is removed when the parallel task chains merge. Finally, the rule associated with task L must not fire until tasks F **and** K have completed.



```
(defrule parallel_rule_B
                ( step A )
=>
                (function 1)
                (function N)
                (assert (next_step B))
)

(defrule parallel_rule_G
                ( step A )
=>
                (function 1)
                (function N)
                (assert (next_step G))
)

(defrule end_decision_rule_L
                ?step1 <- (step F)
                ?step2 <- (step K)
                ?start <- (step A)
=>
                (retract ?step1 ?step2 ?start)
                (function 1)
                (function N)
                (assert (next_step L))
)
```

Figure 16. Parallel Simultaneous Paths

DEVELOPMENT OF CISCO (CENTER INFORMATION SYSTEMS COMPUTER OPERATIONS) ICAT

TARGET is being deployed to acquire knowledge of the mainframe computer operations at JSC. The operating environment being modeled is the IBM VM (Virtual Machine) Operating System on an IBM 3090 super-mainframe CPU. TARGET tracked specific procedures for which operators were responsible. TARGET's ability to transform two-dimensional sequential events into a one-dimensional top-down report was appropriate for the type of tasks facing a mainframe systems operator. Procedures ranged from powering up and down of the CPU and its associated peripheral hardware to the initial program load and shutdown from its console monitor. After TARGET captured procedures from various sources (CIS-B Operator's Manual, SMEs, etc.), task hierarchy reports were generated and verified with the SMEs. This provided the foundation from which an CLIPS rules-based Intelligent Computer-Aided Training ICAT system was to be developed. The following is a description of the strategies taken within TARGET to generate the procedural rules for the CISCO (Center Information System Computer Operations) ICAT knowledge base.

As a procedural KA tool, TARGET models the intricate procedures which an IBM mainframe operator performs as a part of of his/her job. Such tasks as powering up/down mainframe and associated equipment, IPLs (Initial Program Loading), and system shutdowns are captured and converted to a knowledge base that will be used by the CISCO ICAT for mainframe operators at all levels.

Procedures for powering up an IBM mainframe system have been groomed for CLIPS interface. The following actions paired with the relevant translations is displayed in Table 1. Each task or discrete step consists of a specific action and its associated parameters. Through documentation of this action and its components, via the TARGET U.I., the CLIPS rule could be composed. A CLIPS rule requires several pieces of information extracted from a specific step. Figure 17 describes the transformation of a procedural step in the following manner:

Task# 1.1 Verify 208 VAC on voltmeter

Corresponding rule in ICAT format:

```
1    (defrule verify-208-VAC
2                  (step ?s&10)
3                  (environment ? AMD-plr-208-VAC present)
4    =>
5    (assert (message-E-to-I ?s require at-AMD-plr ver-plr VAC-208)
6                  (next-step 20)))
```

Description of the above example:
1 Unique name of rule.
2 Checks the step number.
3 Environment state is checked; *AMD-plr-208-VAC* is the action which contains all the relevant information. The desired state is *present.*
5 Assert *message-E-to-I* is the message from expert to the fact-list. The word *require* signals that this task is a required one. This is followed by an action and a set of parameters. Any parameter may be a combination of an action and parameters.
6 Go to step 20.

Figure 17. Single Task Step with Corresponding CLIPS Rule Content

The succession of task steps (Table 2) from the task hierarchy then becomes associated with CLIPS representations that will run within the CLIPS environment. Each action entry accumulated from the graphical interface will have its associated arguments/states (MOTOR-ON, GEN-ON, RESET-OVERVOLTAGE, GEN-OUTPUT, etc.). As a real application example, CISCO ICAT will use the CLIPS code generated from the TARGET environment to control a learning session concerning the operation and various scenarios within the mainframe computer environment.

	Task Hierarchy of <CIS-B POWER UP>	Domain Expert <Action> and <Arguments> (message-E-to-l <step> require <action> <argument> ...)	Environment State (on LHS) (environment O <variable> <state>)
1.0	3.1.1 AMDAHL Piller Power Up		
1.1	Verify 208 VAC on voltmeter	at-AMD-plr ver-plr 208-VAC	AMD-plr-208-VAC present
1.2	Press RESET OVERVOLTAGE SWITCH	at-AMD-plr prs-plr-sw RESET-OVERVOLTAGE	
1.3	Verify ON/OFF HANDLE (far left) ON (red)	at-AMD-plr ver-plr ON-OFF-HANDLE	
		AMD-plr-ON-OFF-HANDLE ON	
1.4	Press (black) MOTOR ON BUTTON	at-AMD-plr prs-plr-btn MOTOR-ON	
	(grn START LIGHT ON now)	at-AMD-plr ver-plr START-LIGHT	AMD-plr-START-LIGHT ON
1.5	START LIGHT ON > 40 seconds? <NO>		
1.6	Press (black) GENERATOR ON BUTTON	at-AMD-plr prs-plr-btn GEN-ON	AMD-plr-START-LIGHT OFF
1.7	Verify (red) GENERATOR OUTPUT light ON	at-AMD-plr ver-plr GEN-OUTPUT	AMD-plr-GEN-OUTPUT ON
1.8	Verify (red) LOCAL SENSING light ON	at-AMD-plr ver-plr LOCAL-SENSING	AMD-plr-LOCAL-SENSING ON
1.9	Verify two GENERATOR VOLTAGE lights ON	at-AMD-plr ver-plr GEN-VOLTAGE-1	AMD-plr-GEN-VOLTAGE-1 ON
		at-AMD-plr ver-plr GEN-VOLTAGE-2	AMD-plr-GEN-VOLTAGE-2 ON
1.5	START LIGHT ON > 40 seconds? <YES>		
1.10	Press red MOTOR OFF BUTTON	at-AMD-plr prs-plr-btn MOTOR-OFF	AMD-plr-START-LIGHT ON
1.11	Call AMDAHL CE	com-AMD-CE repair-plr	
1.12	Goto 'Verify 208 vac on voltmeter'		

Table 1. Task Actions and their CLIPS Equivalents

Action/Argument	Action Name/Input	# of Args
at-AMD-pLr	Stand at AMDAHL pillar	1
ver-plr	Verify AMDAHL pillar component	1
prs-plr-sw	Press switch	1
prs-plr-btn	Press button	1
com-AMD-CE	Call AMDAHL CE	1
repair-plr	Repair AMDAHL pillar	0
208-VAC	208 VAC on voltmeter	0
RESET-OVERVOLTAGE	RESET OVERVOLTAGE SUITCH	0
ON-OFF-HANDLE	ON/OFF HANDLE at far left	0
MOTOR-ON	Black MOTOR ON BUTTON	0
START-LIGHT	Green START LIGHT	0
GEN-ON	Black GENERATOR ON BUTTON	0
GEN-OUTPUT	Red GENERATOR OUTPUT light	0
LOCAL-SENSING	Red LOCAL SENSING light	0
GEN-VOLTAGE-1	First white GENERATOR VOLTAGE light	0
GEN-VOLTAGE-2	Second white GENERATOR VOLTAGE light	0
MOTOR-OFF	Red MOTOR OFF BUTTON	0

Table 2. Examples of actions and arguments followed by their meaning and number of arguments.

The generated rule conforms to a general ICAT paradigm. However, translation to other knowledge base paradigms or architectures is feasible. The crucial issue is the extraction of actions and parameters from a task. This format is based on a general ICAT architecture developed by the programmers at Computer Sciences Corporation and the Software Technology Branch (PT4) at NASA/Johnson Space Center. The generated rule form is compatible with the ICAT architecture. Further work is being done for rule generation for other paradigms and architectures. The important issue is to extract actions and parameters from a task.

CONCLUSION

The CLIPS world offers TARGET a reasonable paradigm in which to produce knowledge representation from a knowledge acquisition effort. Whereas, TARGET offers the CLIPS world a mechanism in which to generate a knowledge base. As a CLIPS front-end, the system has been able to function as a knowledge engineering mediator for SMEs, programmers, managers and computer novices alike. Whether building a decision tree or a highly complex process network, TARGET provides latitude for a user to document their tasks or jobs with minimal prompting. In addition, the CLIPS code derived from the system would still provide a functional procedural model of the user's world. Overall, TARGET could significantly impact development of various ICAT systems as well as other intelligent systems. For any procedural knowledge acquisition task, it can enhance the ability of the expert to visualize and organize a task or process. Procedural visualization of this type will become more popular as more tools with organizational diagnosis capabilities evolve (Akscyn, McCracken, and Yoder, 1988).

As computer hardware power evolves, more latitude in presentation methods will be available. Visual conception and communication of abstract information will become more common. The strategic fusion of graphical display (bit-map, meta-graphic, etc.) and graphical input device (mouse, light-pen, trackball, etc.) technologies will facilitate visual as well as textual representation of knowledge (Messinger, Rowe, and Henry, 1991). Drawing tools already allow the user to produce and manipulate complex graphics. The role of these tools can also combine with organizational algorithms to create more intelligent diagrams, flow charts and interactive decision trees. With users becoming more adept at employing systems with pictorial modeling capabilities, the mode of procedural KA will also benefit from such advances.

As knowledge acquisition evolves as a discipline within artificial intelligence, more tools to assist in the knowledge acquisition process will also become available in useful forms. TARGET, and tools like it, will be employed within their own "niche" and will also be integrated with other methodologies in the future. Although TARGET currently models the sequence within the task hierarchy structure for rule induction, we will dedicate additional efforts to encapsulating additional knowledge into the steps within a network. In particular, we intend to address issues such as gathering artifact data, selected action rationale, and interactive verification and validation of rules.

REFERENCES

Akscyn, R. M., McCracken, D. L. & Yoder, E. A. (1988). "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations", *Communications of the ACM*, July, 31 (7), 820-834.

Boose, J. H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, March, 1 (1), 3-37.

de Kleer, J., Doyle, J., Steele, G. L., Jr. & Sussman, G. J. (1985). AMORD: Explicit Control of Reasoning. in *Readings in Knowledge Representation*, Brachman, R. J. & Levesque, H. J., Eds., Los Altos, CA: Morgan-Kaufmann Publishers, Inc., 345-355.

Gaines, B. R. (1988). An overview of knowledge-acquisition and transfer. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R & Boose, J. H., Eds., *Knowledge-Based Systems, Vol.1*, New York: Academic Press, 3-22.

Littman, D. C. (1988). Modelling human expertise in knowledge engineering: some preliminary observations. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R. & Boose, J. H., Eds., *Knowledge-Based Systems*, Vol.1, New York: Academic Press, 93-104.

Loftin, R. B., Wang, L., Baffes, L. & Hua, G. (1988). An Intelligent Training System for Space Shuttle Flight Controllers. *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence*, held May 24, 1988, at NASA/Goddard Space Flight Center, Greenbelt, Md, 3-10.

Loftin, R. B., Wang, L., Baffes, P. & Hua, L. (1989). An Intelligent System for Training Space Shuttle Flight Controllers in Satellite Deployment Procedures. *Machine-Mediated Learning*, 3, 43-47.

Messinger, E. B., Rowe, L. A. & Henry, R. R. (1991). A divide-and-conquer algorithm for the layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (1), 1-11.

Payne, S., & Green, T. (1986). Task-action grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, 2, 93-133.